**The 'While' Loop**

Đôi khi bạn muốn lặp lại một hành động trong Ruby trong khi một điều kiện nhất định là đúng, nhưng bạn không biết mình sẽ phải lặp lại hành động đó bao nhiêu lần. Một ví dụ điển hình sẽ là nhắc người dùng về một loại đầu vào nhất định: nếu họ khăng khăng đưa sai cho bạn, bạn có thể phải hỏi lại họ nhiều lần trước khi nhận được loại đầu vào mà bạn đang tìm kiếm.

**The 'Until' Loop**

Phần bổ sung cho whilevòng lặp là untilvòng lặp. Nó giống như một sự lạc hậu while:

i = 0  
until i == 6  
  i = i + 1  
end  
puts i

1. Trong ví dụ trên, đầu tiên chúng ta tạo một biến ivà đặt nó thành 0(không).
2. Sau đó, chúng tôi thực thi một khối mã cho đến khi ibằng 6. Khối mã đó tăng dần i.
3. Khi ibằng 6, khối kết thúc.
4. Cuối cùng, chúng tôi in 6, giá trị của i, vào bảng điều khiển.

Chúng tôi đã sử dụng cú pháp như counter = counter + 1, cú pháp hoạt động, nhưng khi bạn ngày càng thấy với Ruby, luôn có một cách khác.

Một phím tắt là sử dụng một *toán tử gán* . Bạn đã biết một toán tử gán:, =đặt một biến. Bạn có thể cập nhật một biến với toán tử gán thêm, tuy nhiên, trong đó bao gồm +=, -=, \*=, và /=. Ví dụ, khi bạn nhập

counter += 1

Bạn đang nói với Ruby: “Thêm 1 vào counter, sau đó gán lại giá trị mới đó cho counter.” Điều này cung cấp một cách ngắn gọn để cập nhật các giá trị biến trong chương trình của chúng tôi.

Một số ngôn ngữ có các toán tử tăng ++và --(cũng có thể cộng hoặc trừ 1 từ một giá trị), nhưng Ruby thì không. Bạn sẽ phải làm với +=và -=!

**The 'For' Loop**

Bạn nhìn thấy một chút của cú pháp mới trong việc thực hiện theo thời gian: for num in 1...10. Điều này nói với Ruby là: "Đối với biến numtrong phạm vi 1to 10, hãy làm như sau." Sau đây là puts "#{num}", để nummất trên các giá trị từ 1 đến 9, cùng một lúc, những giá trị được in ra cửa sổ Console.

Lý do chương trình này đếm đến 9 chứ không phải 10 là chúng tôi đã sử dụng ba dấu chấm trong phạm vi; điều này yêu cầu Ruby loại trừ số cuối cùng trong số đếm: for num in 1...10có nghĩa là "tăng lên nhưng không bao gồm 10." Nếu chúng ta sử dụng hai dấu chấm, điều này cho biết Ruby bao gồm số cao nhất trong phạm vi.

**The Loop Method**

Cho đến nay, chúng ta đã học được một cách để lặp lại một hành động trong Ruby: sử dụng vòng lặp. Tuy nhiên, như thường lệ trong Ruby, có nhiều cách để hoàn thành một nhiệm vụ nhất định. Trong trường hợp này, cũng có thể lặp lại một hành động bằng cách sử dụng *trình lặp* .

Một trình lặp chỉ là một phương thức Ruby gọi lặp lại một khối mã. Khối mã chỉ là bit chứa các hướng dẫn được lặp lại và những hướng dẫn đó có thể là bất cứ thứ gì bạn thích!

Trình lặp đơn giản nhất là loopphương thức. Bạn có thể tạo một vòng lặp cơ bản (nhưng vô hạn!) Bằng cách chỉ cần gõ

loop { print "Hello, world!" }

Trong Ruby, dấu ngoặc nhọn ( {}) thường có thể hoán đổi cho nhau với các từ khóa do(để mở khối) và end(để đóng nó). Biết được điều này, chúng ta có thể viết một vòng lặp thông minh hơn vòng lặp ở trên:

i = 0  
loop do  
  i += 1  
  print "#{i}"  
  break if i > 5  
end

Các breaktừ khóa là chúng tôi được ra khỏi tù thẻ miễn phí: nó phá vỡ một vòng lặp càng sớm càng điều kiện nó được đáp ứng.

**Next!**

Các nexttừ khóa có thể được sử dụng để bỏ qua bước nào đó trong vòng lặp. Ví dụ: nếu chúng ta không muốn in ra các số chẵn, chúng ta có thể viết:

for i in 1..5  
  next if i % 2 == 0  
  print i  
end

1. Trong ví dụ trên, chúng ta lặp qua phạm vi từ 1 đến 5, gán ilần lượt từng số .
2. Nếu phần còn lại i / 2bằng 0, chúng ta chuyển sang bước lặp tiếp theo của vòng lặp.
3. Sau đó, chúng tôi in giá trị của i. Dòng này chỉ in ra 1, 3và 5vì dòng trước đó.

**The .each Iterator**

Trình looplặp là đơn giản nhất, nhưng cũng là một trong những trình lặp kém mạnh mẽ nhất. Một trình lặp hữu ích hơn là .eachphương thức, có thể áp dụng một biểu thức cho từng phần tử của một đối tượng, tại một thời điểm. Cú pháp có dạng như sau:

object.each { |item|   
  # Do something   
}

Bạn cũng có thể sử dụng dotừ khóa thay cho {}:

object.each do |item|   
  # Do something   
end

Tên biến giữa | |có thể là bất cứ thứ gì bạn thích: nó chỉ là một trình giữ chỗ cho mỗi phần tử của đối tượng mà bạn đang sử dụng .each.

**The .times Iterator**

Các .timesphương pháp giống như một compact siêu forvòng lặp: nó có thể thực hiện một nhiệm vụ trên mỗi mục trong một đối tượng một số quy định của thời đại.

Ví dụ: nếu chúng tôi muốn in ra "Chunky bacon!"mười lần, chúng tôi có thể nhập

10.times { print "Chunky bacon!" }

**Looping with 'While'**

i = 3  
while i > 0 do  
  print i  
  i -= 1  
end

1. Trong ví dụ trên, chúng ta tạo một biến được gọi ivà đặt nó thành 3.
2. Sau đó, chúng tôi in ra 321vì chúng tôi thực hiện vòng lặp miễn ilà tích cực.

**Phương pháp .split**

Tiếp theo, chúng tôi sẽ muốn chia đầu vào của người dùng thành các từ riêng lẻ.

Ruby có một phương thức tích hợp cho cái này được gọi là .split; nó nhận trong một chuỗi và trả về một mảng. Nếu chúng ta chuyển nó một bit văn bản trong dấu ngoặc đơn, .splitsẽ chia chuỗi ở bất cứ nơi nào nó nhìn thấy bit văn bản đó, được gọi là *dấu phân cách* . Ví dụ,

text.split(",")